THE OHIO STATE UNIVERSITY

# Writing a New Automated Testing Method for Custom Display Components

Michael Braun
Kennedy Space Center
Major: Computer Science and Engineering
2019 Spring Session
Date: 05 APR 2019

# Writing a New Automated Testing Method for Custom Display Components

Michael Santana Braun[1]
*John F. Kennedy Space Center, Kennedy Space Center, FL, 32899*

**During the launch process of the Space Launch System (SLS), engineers in the Firing Room of the Launch Control Center (LCC) will be analyzing displays that show data and statistics about the vehicle and launch. These displays are subject to a battery of tests, but the custom components they use are not currently supported by any testing framework. My project is to modify a testing framework so that the custom components on these displays can be tested similarly to how a user would interact with them.**

## Nomenclature

| | |
|---|---|
| NASA | = National Aeronautics and Space Administration |
| KSC | = Kennedy Space Center |
| SLS | = Space Launch System |
| LCS | = Launch Control System |
| LCC | = Launch Control Center |
| CFT | = Combined Functional Test |
| API | = Application Programming Interface |
| GUI | = Graphical User Interface |
| POC | = Point of Contact |

## I. Introduction

During the Spring of 2019 I interned as a software development engineer for the National Aeronautics and Space Administration (NASA) at Kennedy Space Center (KSC). I worked in the Software branch of the Exploration Systems and Operations division on control software to support the ongoing development and launch of the Space Launch System (SLS) series of rockets. To support future launches and associated launch operations, a large suite of display software is being developed for use in the Launch Control Center (LCC), to analyze launch data and manage launch procedures. As a way of ensuring the software's integrity, rigorous testing methods are employed. These include: unit tests, integration tests, and system tests. Recently there has been a push for implementing a new type of test called a Combined Functional Test (CFT), which tests whether displays are updating with expected behavior after actions are performed. My project was to write a new library based on a plugin for a testing framework. This new library would be used to perform CFTs on proprietary components that the plugin could not cover. More specifically, my new library would be able to automatically click on two different types of buttons, read display labels and their properties, and validate images and their properties. Figure 1 shows a diagram of how my code interacts with the existing system.

[1] Spring Intern, NE-XS, NASA Kennedy Space Center, The Ohio State University

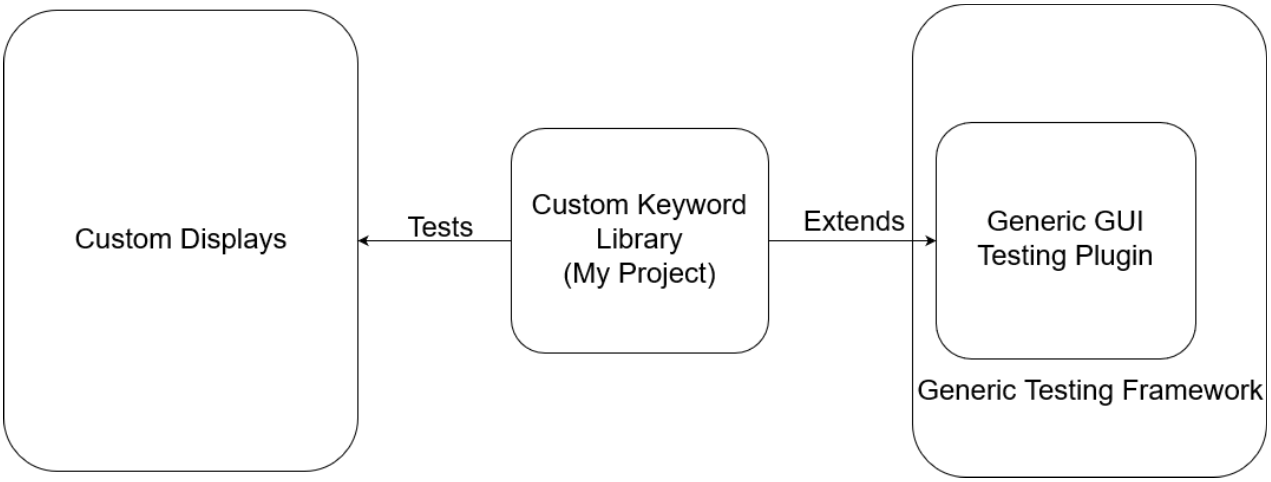


*Figure 1: A block diagram representing my project.*

## II. Approach

### A. Training

The first few weeks of my internship were mostly devoted to setup, training, and familiarization with NASA and its structure. It involved setting up my workstation, requesting and receiving permission to access the various systems and tools I would be using, and meetings to build understanding of the various software components, programs, and acronyms I would need to become familiar with. Once we were all close to having all the required permissions, we met with Jason Kapusta, the Software Architect, to go over the various tasks they needed completed. I was assigned the Display Testing project because it required a relatively deep understanding of an object oriented programming language, a skill that I possessed. While I was training, I read documents about the Display Software and the Dashboard software to familiarize myself with this interface, and understand how each piece would affect our stakeholders.

### B. Using the CFT Testing Framework

At the time of my project, engineers were using a testing framework with a plugin for testing generic Graphical User Interface (GUI) components. Since my project would eventually focus on modifications to the testing framework, I started by reading and running example testing scripts, as well as reading the testing framework documentation. Once I had a decent grasp on the basic testing framework syntax, I moved on to learning about the plugin, and how that would interact with a display, which is shown in Figure 2 below. The conventional method for interacting with custom display components was through mouse movements and screen shots, however, this task required access through an Application Programming Interface (API). Unfortunately, it was unknown if it was even possible to access custom display components through an API. This type of access required using custom test keywords. Although I would eventually be writing custom test keywords, a good interim step would be to see how close I could get to interacting with custom components with the keywords that already existed. I ended up alternating between reading the documentation, using the testing framework's built in reflection to list and call methods for each graphical component and container, and asking my technical Points of Contact (POCs) for advice.

Since the plugin only supports generic display components, I could only navigate to the display and access the top layers of the display. To go deeper into the display hierarchy, I had to access custom containers and then find a path through the different container objects that would allow me to access the buttons and labels on the display. I reviewed the documentation for the custom components to find a suitable path of public methods I could call to access buttons. After a few tries and dead ends, I found a sequence of methods I could use to access every relevant custom component I needed and their parameters. Although much of the testing functionality could be accomplished entirely using the testing framework, a few operations, like pushing the custom buttons, were impossible to accomplish using the stock keywords. When my attempts to push buttons using their publicly accessible parameters only resulted in a graphical change, and did not result in a functional change, I decided I needed to modify the plugin's underlying code to proceed any further.

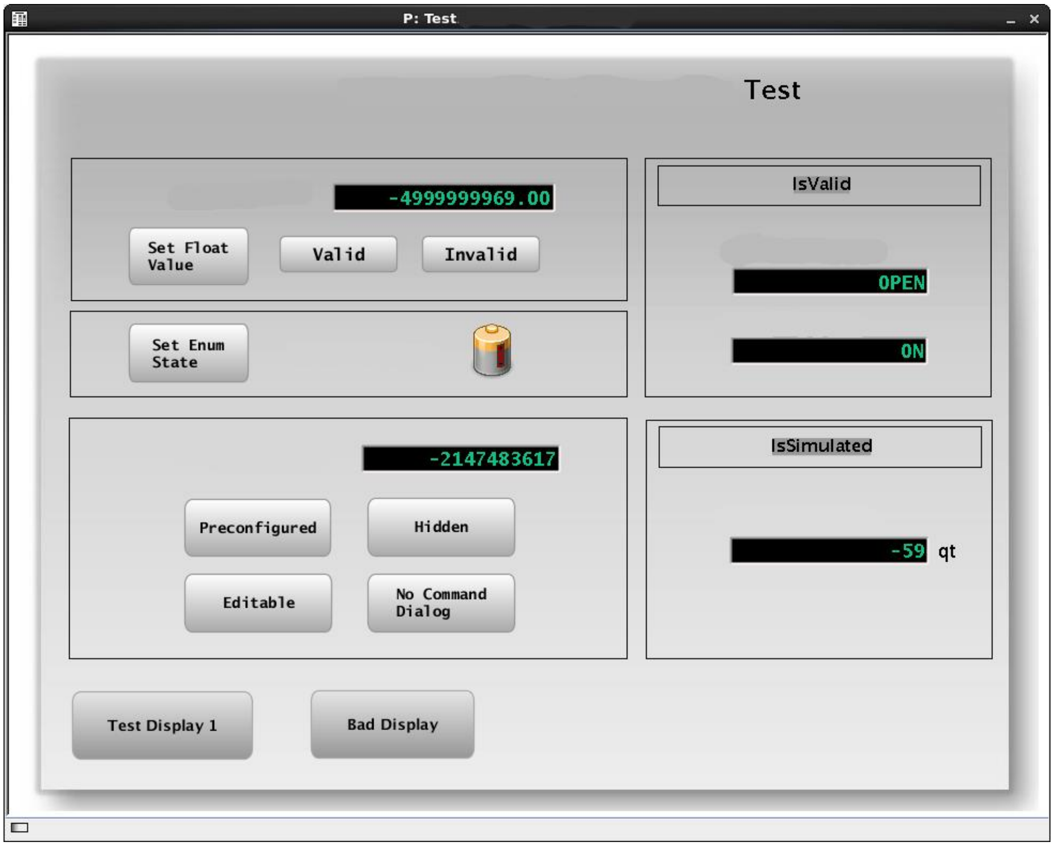


*Figure 2: An example of a test display.*

## C. Translating into Object Oriented Code

Once I had my proof of concept program working in the testing framework, I began work on coding the keywords I needed as an extension of the plugin. This allowed generation of custom objects, which would not be possible normally in the test framework. Mike Owens and Sam Goff, two of the full-time developers, helped me set up a version of the GUI Widget Library on my machine that I could modify and compile myself. Once I learned how to build and replace my own version of the library, I started to recreate the process I used in my script to obtain a list of all the custom components. I added methods that would filter the components based on if the component was a label, text measurement, button, or image.

Next, I worked on methods for button clicks. I wanted to avoid taking mouse control away from the user, so I looked for solutions which would send a mouse click event to a button. I ended up calculating the center of each button from its public width, height, x position, and y position fields. I then sent mouse press and release events to these coordinates on the window to simulate a mouse click. This ended up working well, and once I knew I could get the more difficult button clicking behavior to work, I started working on implementing the easier but repetitive methods.

## D. Communicating Test Requirements

One of the bigger technical roadblocks I had to overcome during this project was how to make my library both useful and useable for the testers. This meant that some of what appeared to be insignificant details could become areas that needed input from multiple perspectives to determine an optimal solution. One such area involved uniquely identifying custom components. The only completely unique value was a hash value that was inaccessible to testers, and a component id that was automatically generated that I was unable to access. Since both methods could break if components were updated on the display, we started to consider methods that did not use perfectly unique keys but had the advantage of being more reliable and less likely to break.

Each custom component is associated with a user-facing identifier. However, that identifier may not be unique on the display. After discussing this identifier as a possible key with other stakeholders in the project, I decided on using it as the primary key for some types of objects where it doesn't make sense to have duplicates with the same identifier. For other types of objects, like the image, we determined that the identifier would not be enough for most cases and added another overloaded selection option that would select by the identifier and an image name.

Next, the number of keywords necessary for testers to work effectively, and which ones would be important to focus my energy on needed to be determined. After discussing with members of my team, I decided to stick with keywords for reading all the parameters from each component, identifying components and pushing buttons, since those keywords would be the core of my project, and cover most cases where testers would use them. After implementing most of the keywords, and discussing it more with my team, I started the code review process and got feedback about my code quality.

## III. Conclusion

### A. Summary

My project during this internship will have a significant impact for those who are working on the command and control displays. Its impact also extends to the engineers who will actually be using the displays and, by extension, to the safety of SLS. My project will make it possible to robustly automate tests on the display, and thus make it easier and more efficient for testers to write and run tests multiple times. The time saved testing will be invaluable to the test engineers and will allow coding issues to be resolved earlier and more efficiently.

### B. Future Developments

There are many useful features for my project that may be implemented in the future to fulfill the needs of various teams who use this method of testing. For example, there are normally only four types of custom components that most teams use, but other teams can create and add their own custom components to supplement the main four. These new components will need their own functions written to access their specific properties but can use my code as a basis for their development. Enhancements such as the addition of predefined colors to the codebase, so testers can identify the color of a component without having to know the specific RGB color code are also something that may be considered in the future.

## Acknowledgments

I would like to thank my mentors, Jill Giles and Jamie Szafran, for all the help and guidance they gave me throughout my internship and for always being excited to talk to me and share their knowledge. Special thanks to Sam Goff and Kathy Saint for helping me figure out the solutions and configurations that I would never have found on my own, as well as spending time to explain how the software worked and pointing out any resources that could help me out. Thank you to Jason Kapusta and Tony Ciavarella for setting up my project, giving me a strong direction and answering any tough questions I had confidently. I'd like to thank Mike Hewitt, Ricky Ludwig, Mike Owens and all others who helped me get past the obstacles in my way. I'd also like to thank Gwen Gamble and the KSC Education office for setting up this internship, all the work they did to support us along the way, and for setting up all the interesting tours and intern events.

## References

[1] Ohio State University Logo

https://brand.osu.edu/logo/